ICIW 2006, Guadeloupe
Adding semantics to Web services with the Web Service Modeling Ontology

Mick Kerrigan
Jacek Kopecky
Matthew Moran
Dumitru Roman
Brahmananda Sapkota
Liliana Cabral
John Domingue
Stefania Galizia
Barry Norton
The **aims** of this tutorial

- Introduce the aims & challenges of Semantic Web Services (SWS) - the WSMO approach
- Present a general overview of a fully fledged framework for SWS: a conceptual model, a language, and execution environments
- Experience and work with WSMO enabled tools and systems
But first a few words about us…

• We are members of:
  – Knowledge Media Institute (KMi) (at Open University) – Conversational Hypermedia, GroupWare, Telepresence, Knowledge Management in Engineering, Knowledge Engineering for Narrative Creation, Semantic Web and Knowledge Services
  – Digital Enterprise Research Institute (DERI) - DERI’s vision is to make the Semantic Web and Semantic Web Services a reality enabling fully flexible eCommerce for small, medium-sized and large enterprises.

• Our main focus - Semantic Web Services; SWS have the potential to become a key-enabling infrastructure for Knowledge Management and eWork, Enterprise Application Integration, and eCommerce

=> In consequence, Semantic Web Services are one of the key areas of applied computer science
Major **technologies** currently developed by DERI & KMl (in cooperation with other institutions)

- **WSMO** - an ontology for Semantic Web Services
- **WSML** - Semantic Web Services and Semantic Web languages
- **WSMX** - an execution environment for Semantic Web Services compliant with WSMO/L
- **Triple Space Computing** - communication platform for Semantic Web services based on Web principles: “Persistently publish and read semantic data that is denoted by unique identifiers”
- **IRS** - Semantic Web Services framework

**In the focus of this tutorial**
# Agenda

## Part I: Introduction to Semantic Web Services; Concepts and Languages – the WSMO perspective

<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
<th>Presenters</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00 – 12:15</td>
<td>Part I: Introduction to Semantic Web Services; Concepts and Languages – the WSMO perspective</td>
<td>Jacek Kopecky, Dumitru Roman</td>
</tr>
</tbody>
</table>

## Lunch

<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:30 – 14:00</td>
<td>Lunch</td>
</tr>
</tbody>
</table>

## Part II: WSMO enabled systems and tools; hands-on sessions

<table>
<thead>
<tr>
<th>Time</th>
<th>Topic</th>
<th>Presenters</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:00 – 17:00</td>
<td>Part II: WSMO enabled systems and tools; hands-on sessions</td>
<td>Stefania Galizia, Barry Norton, Brahmananda Sapkota</td>
</tr>
</tbody>
</table>
Part I: Introduction to Semantic Web Services; Concepts and Languages – the WSMO perspective
## Part I - Agenda

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00 – 10:30</td>
<td>Introduction to Semantic Web Services</td>
</tr>
<tr>
<td></td>
<td>Web Service Modelling Ontology (WSMO)</td>
</tr>
<tr>
<td>10:30 – 11:00</td>
<td>Coffee Break</td>
</tr>
<tr>
<td>11:00 – 12:30</td>
<td>Web Service Modeling Language (WSML)</td>
</tr>
<tr>
<td></td>
<td>WSMO Discovery</td>
</tr>
<tr>
<td></td>
<td>WSMO Grounding</td>
</tr>
</tbody>
</table>
Intro to Semantic Web Services

• Introduction to Semantic Web

• Introduction to Web services

⇒ Semantic Web Services
Semantic Web - The Vision

- 500 million users
- more than 3 billion pages

Syntax:
- URI, HTML, HTTP

Static

Dynamic

WWW
Semantic Web - The Vision

Serious Problems with
- finding,
- extraction,
- representation,
- interpretation and
- maintenance
of information

WWW
URI, HTML, HTTP

→ Semantic Web
RDF, RDF(S), OWL

Dynamic

Static

Syntax Semantics
Semantic Web - The Vision

Dynamic

Web Services
SOAP, WSDL, UDDI

Static

WWW
URI, HTML, HTTP

Semantic Web
RDF, RDF(S), OWL

Syntax

Semantics

Bringing the computer back as a device for computation
Semantic Web - The Vision

Bringing the web to its full potential

Dynamic

Web Services
- UDDI, WSDL, SOAP

Intelligent Web Services

Static

WWW
- URI, HTML, HTTP

Semantic Web
- RDF, RDF(S), OWL

Syntax

Semantics
Ontology Definition

- formal, explicit specification of a shared conceptualization
  - unambiguous definition of all concepts, attributes and relationships
  - conceptual model of a domain (ontological theory)
  - machine-readability
  - commonly accepted understanding
Ontology Example

Concept
contectual entity of the domain

Attribute
property of a concept

Relation
relationship between concepts or properties

Axiom
coherent description between Concepts / Properties / Relations via logical expressions

holds(Professor, Lecture) \Rightarrow \text{Lecture.topic} \in \text{Professor.researchField}
Ontology Languages

• Requirements:
  – "expressivity"
    • knowledge representation
    • ontology theory support
  – "reasoning support"
    • sound (unambiguous, decidable)
    • support of reasoners / inference engines

• Semantic Web languages:
  – web compatibility
  – Existing W3C Recommendations:
    • XML, RDF, OWL
Semantic Web Language Layer Cake
Web Services

Web Services: [Stencil Group]
• loosely coupled, reusable components
• encapsulate discrete functionality
• distributed
• programmatically accessible over standard internet protocols
• add new level of functionality on top of the current web
Using Web Services
Using Web Services

- UDDI Registry
  - Points to Description
  - Finds Service
- WSDL
  - Describes Service
- Service Consumer
  - SOAP
    - Communicates with XML Messages
- Web Service
Lack of SWS standards

Current technology does not allow realization of any of the parts of the Web Service usage process:

- Only syntactical standards available
- Lack of fully developed semantic markup languages
- Lack of semantically marked up content and services
- Lack of semantically enhanced repositories
- Lack of frameworks that facilitate discovery, composition and execution
- Lack of tools and platforms that allow to semantically enrich current Web content
Semantic Web Services

- Define exhaustive description frameworks for describing Web Services and related aspects (Web Service Description Ontologies)
- Support ontologies as underlying data model to allow machine supported data interpretation (Semantic Web aspect)
- Define semantically driven technologies for automation of the Web Service usage process (Web Service aspect)
Semantic Web Services (2)

Usage Process:

• **Publication**: Make available the description of the capabilities of a service
• **Discovery**: Locate different services suitable for a given task
• **Selection**: Choose the most appropriate services among the available ones
• **Composition**: Combine services to achieve a goal
• **Mediation**: Solve mismatches (in data or process) among the combined services
• **Execution**: Invoke services following programmatic conventions
Semantic Web Services (3)

Usage Process – execution support

- **Monitoring:** Control the execution process
- **Compensation:** Provide transactional support and undo or mitigate unwanted effects
- **Replacement:** Facilitate the substitution of services by equivalent ones
- **Auditing:** Verify that service execution occurred in the expected way
Summary

Semantic Web Services
= Semantic Web Technology
+ Web Service Technology
Web Service Modeling Ontology (WSMO)

• A conceptual model for Semantic Web Services:
  – Ontology of core elements for Semantic Web Services
  – a formal description language (WSML)
  – execution environment (WSMX)

• … derived from and based on the Web Service Modeling Framework WSMF

• an SDK-Cluster Working Group
  (joint European research and development initiative)
WSMO Working Groups

A Conceptual Model for SWS

A Formal Language for WSMO

A Rule-based Language for SWS

Execution Environment for WSMO
WSMO Design Principles

- Web Compliance
- Ontology-Based
- Strict Decoupling Of Modeling Elements
- Ontological Role Separation
- Centrality of Mediation
- Execution Semantics
- Description versus Implementation
Objectives that a client wants to achieve by using Web Services

- Provide the formally specified terminology of the information used by all other components
- Semantic description of Web Services:
  - **Capability** (functional)
  - **Interfaces** (usage)
- Connectors between components with mediation facilities for handling heterogeneities

**WSMO D2, version 1.2, 13 April 2005 (W3C submission)**
Non-Functional Properties

every WSMO elements is described by properties that contain relevant, non-functional aspects

- Dublin Core Metadata Set:
  - complete item description
  - used for resource management
- Versioning Information
  - evolution support
- Quality of Service Information
  - availability, stability
- Other
  - Owner, financial
## Non-Functional Properties List

<table>
<thead>
<tr>
<th>Dublin Core Metadata</th>
<th>Quality of Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contributor</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Coverage</td>
<td>NetworkRelatedQoS</td>
</tr>
<tr>
<td>Creator</td>
<td>Performance</td>
</tr>
<tr>
<td>Description</td>
<td>Reliability</td>
</tr>
<tr>
<td>Format</td>
<td>Robustness</td>
</tr>
<tr>
<td>Identifier</td>
<td>Scalability</td>
</tr>
<tr>
<td>Language</td>
<td>Security</td>
</tr>
<tr>
<td>Publisher</td>
<td>Transactional</td>
</tr>
<tr>
<td>Relation</td>
<td>Trust</td>
</tr>
<tr>
<td>Rights</td>
<td>Other</td>
</tr>
<tr>
<td>Source</td>
<td>Financial</td>
</tr>
<tr>
<td>Subject</td>
<td>Owner</td>
</tr>
<tr>
<td>Title</td>
<td>市委常委</td>
</tr>
<tr>
<td>Type</td>
<td>TypeOfMatch</td>
</tr>
<tr>
<td><strong>Non-Functional Properties List</strong></td>
<td><strong>Quality of Service</strong></td>
</tr>
<tr>
<td><strong>Dublin Core Metadata</strong></td>
<td><strong>Accuracy</strong></td>
</tr>
<tr>
<td><strong>Contributor</strong></td>
<td><strong>NetworkRelatedQoS</strong></td>
</tr>
<tr>
<td><strong>Coverage</strong></td>
<td><strong>Performance</strong></td>
</tr>
<tr>
<td><strong>Creator</strong></td>
<td><strong>Reliability</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Robustness</strong></td>
</tr>
<tr>
<td><strong>Format</strong></td>
<td><strong>Scalability</strong></td>
</tr>
<tr>
<td><strong>Identifier</strong></td>
<td><strong>Security</strong></td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td><strong>Transactional</strong></td>
</tr>
<tr>
<td><strong>Publisher</strong></td>
<td><strong>Trust</strong></td>
</tr>
<tr>
<td><strong>Relation</strong></td>
<td><strong>Other</strong></td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td><strong>Financial</strong></td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td><strong>Owner</strong></td>
</tr>
<tr>
<td><strong>Subject</strong></td>
<td><strong>常委</strong></td>
</tr>
<tr>
<td><strong>Title</strong></td>
<td><strong>TypeOfMatch</strong></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><strong>Version</strong></td>
</tr>
<tr>
<td><strong>Non-Functional Properties List</strong></td>
<td><strong>Quality of Service</strong></td>
</tr>
</tbody>
</table>
WSMO Ontologies

Objectives that a client wants to achieve by using Web Services

- Capability (functional)
- Interfaces (usage)

Provide the formally specified terminology of the information used by all other components

Connectors between components with mediation facilities for handling heterogeneities
Ontology Usage & Principles

- **Ontologies are used as the ‘data model’ throughout WSMO**
  - all WSMO element descriptions rely on ontologies
  - all data interchanged in Web Service usage are ontologies
  - Semantic information processing & ontology reasoning

- **WSMO Ontology Language WSML**
  - conceptual syntax for describing WSMO elements
  - logical language for axiomatic expressions (WSML Layering)

- **WSMO Ontology Design**
  - Modularization: import / re-using ontologies, modular approach for ontology design
  - De-Coupling: heterogeneity handled by **OO Mediators**
Ontology Specification

- **Non functional properties**  (see before)
- **Imported Ontologies**  importing existing ontologies where no heterogeneities arise
- **Used mediators**  OO Mediators (ontology import with terminology mismatch handling)

**Ontology Elements:**
- **Concepts**  set of concepts that belong to the ontology, incl.
- **Attributes**  set of attributes that belong to a concept
- **Relations**  define interrelations between several concepts
- **Functions**  special type of relation (unary range = return value)
- **Instances**  set of instances that belong to the represented ontology
- **Axioms**  axiomatic expressions in ontology (logical statement)
WSMO Web services

Objectives that a client wants to achieve by using Web Services

Provide the formally specified terminology of the information used by all other components

Connectors between components with mediation facilities for handling heterogeneities

Semantic description of Web Services:
- **Capability** (functional)
- **Interfaces** (usage)
WSMO Web service description

Non-functional Properties
- complete item description
- quality aspects
- Web Service Management

Capability
- Advertising of Web Service
- Support for WS Discovery

DC + QoS + Version + financial

Web service implementation

(realization of functionality by aggregating other Web Services)
- functional decomposition
- WS composition

Choreography --- Service Interfaces --- Orchestration

client-service interaction interface for consuming WS
- External Visible Behavior
- Communication Structure
- 'Grounding'

functional description
Capability Specification

- **Non functional properties**
- **Imported Ontologies**
- **Used mediators**
  - *OO Mediator*: importing ontologies with mismatch resolution
  - *WG Mediator*: link to a Goal wherefore service is not usable a priori
- **Pre-conditions**
  - What a web service expects in order to be able to provide its service. They define conditions over the input.
- **Assumptions**
  - Conditions on the state of the world that has to hold before the Web Service can be executed
- **Post-conditions**
  - Describes the result of the Web Service in relation to the input, and conditions on it
- **Effects**
  - Conditions on the state of the world that hold after execution of the Web Service (i.e. changes in the state of the world)
Choreography & Orchestration

• **VTA example:**

  - **Choreography** = how to interact with the service to consume its functionality
  - **Orchestration** = how service functionality is achieved by aggregating other Web services
Choreography Aspects

*Interface for consuming Web Service*

- **External Visible Behavior**
  - those aspects of the workflow of a Web Service where Interaction is required
  - described by workflow constructs: sequence, split, loop, parallel

- **Communication Structure**
  - messages sent and received
  - their order (communicative behavior for service consumption)
  - choreography related errors (e.g. input wrong, message timeout, etc.)

- **Grounding**
  - concrete communication technology for interaction

- **Formal Model**
  - reasoning on Web Service interfaces (service interoperability)
  - allow mediation support on Web Service interfaces
Orchestration Aspects

Control Structure for aggregation of other Web Services

- decomposition of service functionality
- all service interaction via choreographies
Orchestration Aspects

• Service interfaces are concerned with service consumption and interaction
• Choreography and Orchestration as sub-concepts of Service Interface
• Common requirements for service interface description:
  1. represent the dynamics of information interchange during service consumption and interaction
  2. support ontologies as the underlying data model
  3. appropriate communication technology for information interchange
  4. sound formal model / semantics of service interface specifications in order to allow operations on them.
**Future Directions**

**Choreography:**
- interaction of services / service and client
- a „choreography interface“ describes the behavior of a Web Service for client-service interaction for consuming the service

**Orchestration:**
- how the functionality of a Web Service is achieved by aggregating other Web Services
- extends Choreography descriptions by control & data flow constructs between orchestrating WS and orchestrated WSs.

**Ontologies as data model:**
- every resource description based on ontologies
- every data element interchanged is ontology instance

**Formal description of service interfaces:**
- ASM-based approach
- allows reasoning & mediation

**User language**
- based on UML2 activity diagrams
- graphical Tool for Editing & Browsing Service Interface Description

**Grounding:**
- making service interfaces executable
- currently grounding to WSDL

**Workflow constructs as basis for describing service interfaces:**
- workflow based process models for describing behavior
- on basis of generic workflow constructs (e.g. van der Aalst)

**Conceptual models**
WSMO Goals

Objectives that a client wants to achieve by using Web Services

- Provide the formally specified terminology of the information used by all other components

Ontologies

Goals

Web Services

Mediators

Connectors between components with mediation facilities for handling heterogeneities

Semantic description of Web Services:
- **Capability** (functional)
- **Interfaces** (usage)
Goals

- **Ontological De-coupling of Requester and Provider**

- **Goal-driven Approach**, derived from AI rational agent approach
  - Requester formulates objective independently
  - ‘Intelligent’ mechanisms detect suitable services for solving the Goal
  - allows re-use of Services for different purposes

- **Usage of Goals within Semantic Web Services**
  - A Requester, that is an agent (human or machine), defines a Goal to be resolved
  - Web Service Discovery detects suitable Web Services for solving the Goal automatically
  - Goal Resolution Management is realized in implementations
Goal Specification

- Non functional properties
- Imported Ontologies
- Used mediators
  - **OO Mediators**: importing ontologies with heterogeneity resolution
  - **GG Mediator**:
    - Goal definition by reusing an already existing goal
    - allows definition of **Goal Ontologies**
- Requested Capability
  - describes service functionality expected to resolve the objective
  - defined as capability description from the requester perspective
- Requested Interface
  - describes communication behaviour supported by the requester for consuming a Web Service (Choreography)
  - Restrictions / preferences on orchestrations of acceptable Web Services
WSMO Mediators

Objectives that a client wants to achieve by using Web Services

- Capability (functional)
- Interfaces (usage)

Provide the formally specified terminology of the information used by all other components

Connectors between components with mediation facilities for handling heterogeneities
Mediation

• **Heterogeneity** …  
  – Mismatches on structural / semantic / conceptual / level  
  – Occur between different components that shall interoperate  
  – Especially in distributed & open environments like the Internet

• **Concept of Mediation** (Wiederhold, 94):  
  – *Mediators* as components that resolve mismatches  
  – **Declarative Approach:**  
    • Semantic description of resources  
    • ‘Intelligent’ mechanisms that resolve mismatches independent of content  
  – Mediation cannot be fully automated (integration decision)

• **Levels of Mediation within Semantic Web Services** (WSMF):  
  (1) **Data Level:** mediate heterogeneous Data Sources  
  (2) **Protocol Level:** mediate heterogeneous Communication Patterns  
  (3) **Process Level:** mediate heterogeneous Business Processes
WSMO Mediators Overview
Mediator Structure

Source Component

1..n

WSMO Mediator

uses a Mediation Service via

Target Component

- as a Goal
- directly
- optionally incl. Mediation

Mediation Services
Merging 2 ontologies

Goal:
“merge s1, s2 and s1.ticket subclassof s2.product”

OO Mediator
Mediation Service

Discovery

Mediation Services

Train Connection Ontology (s1)

Purchase Ontology (s2)

Train Ticket Purchase Ontology

OO Mediator - Example
GG Mediators

- **Aim:**
  - Support specification of Goals by re-using existing Goals
  - Allow definition of **Goal Ontologies** (collection of pre-defined Goals)
  - Terminology mismatches handled by OO Mediators

- **Example: Goal Refinement**

```
Source Goal
“Buy a ticket”

GG Mediator
Mediation Service

Target Goal
“Buy a Train Ticket”

postcondition:
“aTicket memberof trainticket”
```
WG & WW Mediators

- **WG Mediators:**
  - link a Web Service to a Goal and resolve occurring mismatches
  - match Web Service and Goals that do not match a priori
  - handle terminology mismatches between Web Services and Goals
  ⇒ broader range of Goals solvable by a Web Service

- **WW Mediators:**
  - enable interoperability of heterogeneous Web Services
  ⇒ support automated collaboration between Web Services

- **OO Mediators** for terminology import with data level mediation
- Protocol Mediation for establishing valid multi-party collaborations
- Process Mediation for making Business Processes interoperable
Web Service Modeling Language (WSML)

- Aim – to provide a language (or a set of interoperable languages) for representing the elements of WSMO:
  - Ontologies, Web services, Goals, Mediators
- WSML provides a formal language for the conceptual elements of WSMO, based on:
  - Description Logics
  - Logic Programming
  - First-Order Logic
  - Frame Logic
Rationale of WSML

• Provide a Web Service Modeling Language based on the WSMO conceptual model
  – Concrete syntax
  – Semantics
• Provide a Rule Language for the Semantic Web
• Many current Semantic Web languages have
  – undesirable computational properties
  – unintuitive conceptual modeling features
  – inappropriate language layering
    • RDFS/OWL
    • OWL Lite/DL/Full
    • OWL/SWRL
Variants of WSML

- WSML-DL
  - First-Order Logic
    (with nonmonotonic extensions)
  - Description Logics
- WSML-Core
- WSML-Flight
- WSML-Rule

Logic Programming
WSML-Core

• Basic interoperability layer between Description Logics and Logic Programming paradigms
• Based on Description Logic Programs
  – Expressive intersection of Description Logic SHIQ and Datalog
  – Allows to take advantage of many years of established research in Databases and Logic Programming
  – Allows reuse of existing efficient Deductive Database and Logic programming reasoners
• Some limitations in conceptual modeling of Ontologies
  – No cardinality constraints
  – Only “inferring” range of attributes
  – No meta-modeling
WSML-DL

• Extension of WSML-Core
• Based on the Description Logic SHIQ
  – Entailment is decidable
  – Close to DL species of Web Ontology Language OWL
  – Many efficient subsumption reasoners
• Some limitations in conceptual modeling of Ontologies
  – No cardinality constraints
  – Only “inferring” range of attributes
  – No meta-modeling
• Limitations in logical expressions
  – From Logic Programming point-of-view, there is a lack of:
    • N-ary predicates
    • Chaining variables over predicates
    • (Default) negation
WSML-Flight

• Extension of WSML-Core
• Based on the Datalog,
  – Ground entailment is decidable
  – Allows to take advantage of many years of established research in Databases and Logic Programming
  – Allows reuse of existing efficient Deductive Database and Logic programming reasoners
• No limitations in conceptual modeling of Ontologies
  – Cardinality constraints
  – Value constraints for attributes
  – Meta-modeling
WSML-Rule

• Extension of WSML-Flight; based on Horn fragment of F-Logic
  – Ground entailment is undecidable
  – Turing complete
  – Allows to take advantage of many years of established research in Logic Programming
  – Allows reuse of existing efficient Logic programming reasoners
• Extends WSML-Flight logical expressions with:
  – Function symbols
  – Unsafe rules
• From Description Logic point-of-view, there is a lack of:
  – Existentials
  – Disjunction
  – (Classical) negation
  – Equality
WSML-Full

- Extension of WSML-Rule and WSML-DL
- Based on First Order Logic with nonmonotonic extensions
  - Entailment is undecidable
  - Very expressive
- Extends WSML-DL logical expressions with:
  - Chaining variables over predicates
  - Function symbols
  - Nonmonotonic negation
  - N-ary predicates
- Extends WSML-Rule with:
  - Existentials
  - Disjunction
  - Classical negation
  - Equality
- Specification of WSML-Full is open research issue
WSML - example

wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace { _"http://www.example.org/example#”, dc _”http://purl.org/dc/elements/1.1/”}

ontology _”http://www.example.org/exampleOntology”
 [...]

goal _”http://www.example.org/exampleGoal”
 [...]

etc...
WSML Syntax

- WSML human-readable syntax
- WSML exchange syntaxes:
  - XML syntax:
    - Syntax for exchange over the Web
    - Translation between human-readable and XML syntax
    - XML Schema for WSML has been defined
  - RDF syntax:
    - Interoperability with RDF applications
    - Maximal reuse of RDF and RDFS vocabulary
    - WSML RDF includes most of RDF
    - Translation between human-readable and RDF syntax
    - For logical expressions, XML literals are used
WSMO Discovery

- Web Service vs. Service
- Automated WS discovery
- Descriptions and Discovery
- WSMO Discovery process
Web Service vs. Service

• Notions of **Web Service** & **Service** are often interpreted in various ways in the literature
• We use the following **terminology & interpretation** here
  – **Service**
    • A **provision of value in some domain** (not necessarily monetary, independent of how service provider and requestor interact)
  – **Web Service**
    • **Computational entity** accessible over the Internet (using Web Service Standards & Protocols), **provides access to (concrete) services** for the clients.

• Thus, we have the following **relation between the notions**:
  – **Service** corresponds to a **concrete execution of a Web service** (with given input values)
  – **Web Service** provides a **set of services** to its client; one service for each possible input value tuple
Automated WS discovery

• The task
  – Identify possible web services \( W \) which are able to provide the requested service \( S \) for ist clients

• An important issue …
  – „being able to provide a service“ has to be determined based on given descriptions only (\( WS, \) Goal, Ontos)
  – Discovery can only be as good as these descriptions
    • Very detailed WS descriptions: are precise, enable highly accurate results, are more difficult to provide; in general, requires interaction with the provider (outside the pure logics framework)
    • Less detailed WS descriptions: are easy to provide for humans, but usually less precise and provide less accurate results
Descriptions and Discovery (I)

- We aim at **supporting a wide-variety** of clients and applications
  - Support different description techniques for clients
  - Support a wide-variety of applications wrt. needed accuracy
  - **Main focus here**: Capability – What does the service deliver?

- **Basic possibilities for the description of web services**:

  - **Syntactic approaches**
    - Keyword-based search, natural language processing techniques, Controlled vocabularies
  
  - **Lightweight semantic approaches**
    - Ontologies, What does W provide (not how)?, Action-Object-Modelling, Coarse-grained semantic description of a service

  - **Heavyweight semantic approaches**
    - Describes the service capability in detail, Pre/Post-Cond, takes „in-out“ relationship into account, Fine-grained web service description
Descriptions and Discovery (II)

- **Service provider side:**
  - Capability description & levels of abstraction

What do I provide?
  - (Syntactically)
  - (Semantically)
  - When (for what input)?

{Keyword}

WS

- Syntactic
  - Semantic ("Light")
  - Semantic ("Heavy")
Descriptions and Discovery (III)

- **Service requester side**: Goal description

  - **What do I want?** (Syntactically)
    - \{Keyword\} K1 ... Kn
  
  - **What do I want?** (Semantically)
  
  - **What do I want & What (input) can I provide?** (Semantically)

  - **Level of Abstraction**
    - Syntactic
    - Semantic ("Light")
    - Semantic ("Heavy")
Descriptions and Discovery (IV)

- Basic idea for Matching on the single levels

Common keywords

Set-theoretic relationship

Adequate (common) execution/state-transition

{Keyword}

W1 ... WL K1 ... Kn

WS

Syntactic

Semantic ("Light")

Semantic ("Heavy")
Descriptions and Discovery (V)

- Capability descriptions: **Layers of Capabilities**
  - How to combine various levels of abstraction?
Descriptions and Discovery (VI)

• Capability descriptions:
  – Levels of abstraction & possible accuracy?

What? (Syntactically)
  → Syntactic capability
    perhaps complete & perhaps correct

What? (Semantically)
  → Abstract capability
    complete & perhaps correct

What & When? (Semant.)
  → Concrete capability
    complete & correct (if user input known & interaction)
Descriptions and Discovery (VII)

• Possible approaches for checking matches and their assumed costs

Information Retrieval: efficient
DL-based reasoning/deductive databases: more or less efficient
Deductive databases with TA-Logic support/ Theorem-Proving: less efficient/no guarantees

Level of Abstraction

Syntactic
Semantic („Light“)
Semantic („Heavy“)
**Keyword-based description and discovery**

- Service descriptions and user request: bag of keywords
- Simple syntactic matching
- Uses relevant keywords for matching: NFP values, etc.
“Lightweight” descriptions and discovery

- **Service providing a value in some domain:**
  - Goal describes the desired post state as a set of objects
  - Service describes the state after its execution

- **Intentions:**
  - Describe if the Requester/Provider requests/provides all objects or just one of the objects in the set
“Heavyweight” descriptions and discovery

- **Web Service as a computational entity**
  - Takes input values $I_1, \ldots, I_n$ that fulfill certain properties (precondition)
  - Input values determine Outputs $O(I_1, \ldots, I_n)$ and Effects $E(I_1, \ldots, I_n)$

- **Semantics**
  - Web Service as a state-relation (transformation)
  - Captured by:
    - Precondition/Assumptions
    - Postcondition/Effects
WSMO Discovery Process (I)

- **Distinguish further between**
  - Web Service Discovery
  - Service Discovery
- **Web Service Discovery**
  - No interaction with the provider, matches are only based on **static capability descriptions**
  - Matching is less accurate (we can only return web services which might be able to deliver a requested service)
  - Possibly ignore preconditions and inputs in service capabilities
  - Most likely with abstract capabilities
- **Service Discovery**
  - Interaction with the provider with concrete input from user (**dynamic capabilities**)
  - Only with heavyweight descriptions of service capabilities possible (Input has to be considered!)
  - Matching is can be as accurate as possible
  - The more interaction, the less efficient becomes checking a match
WSMO Discovery Process (II)

The process envisioned at present …

Goal-Repos.
- Predefined formal Goal

Available WS
- Abstract Capability

Requester Desire
- Goal Discovery
- Selected predefined Goal

Requester Goal
- Goal refinement

Web Service Discovery
- Concrete Capability (possibly dynamic)
- Still relevant WS

Web Service (Service Discovery)
- Service to be returned
WSMO Grounding

• Motivation
  – It’s a WSDL and XML Schema world

• Background
  – XML, XML Schema, what’s been done before

• Approaches to data grounding
  – Three possible approaches, one chosen

• Creating the mappings
  – Methodology, identifying mappings, next steps

• Grounding WSMO Choreography
Motivation

• Web services being created and deployed now and for the next few years will be described using WSDL and XML Schema
• Want to define the mechanism for how WSMO service descriptions can be grounded to WSDL
  – Ground WSMO ontologies to XML Schema
  – Ground WSMO choreography descriptions to WSDL operations
Background – XML

• Standard language for sharing data across systems, especially on the Web
• Application-dependent tag set → great flexibility
• Many XML based languages for all kinds of purposes
• Strong tool support: parsers, editors, storage, querying

• Semantics must be known by receiver in advance; can not be determined from the document itself
Background – XML Schema

- Defines the structure of XML documents
  - Legal elements and attributes
  - Order, cardinalities of child elements
  - Default and fixed values for elements and attributes

- Components of XML Schema
  - Element declarations (global or local)
  - Attribute declarations (global or local)
  - Simple types
    - Built-in or defining constraints on values of built-in types
  - Complex types
    - For elements
    - Define attributes, child elements
    - Extend or restrict definition of an existing complex type
Background – Previous Work

Comparing XML schema languages (DTD, XS) to Ontologies

<table>
<thead>
<tr>
<th>XML schema language</th>
<th>Ontologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define vocabulary and constraints for XML docs</td>
<td>Formal specification of shared domain theory</td>
</tr>
<tr>
<td>Structure</td>
<td>Meaning, no explicit structure</td>
</tr>
</tbody>
</table>

Other Related Areas of Work

- Embedding semantic metadata into XML
  - Complement structure with semantics
- **Lifting** XML representation to OWL and RDF
  - We will take a similar approach
- **Lowering** ontologies to XML schema
  - More expressive to less expressive
Approaches to Mapping

Ontology from schema

instance _1 memberOf Person
  name hasValue _2
  instance _2 memberOf Name
    firstName hasValue "John"
    lastName hasValue "Doe"

Direct mapping language

source XML data

XSLT

<instance>
  <memberOf>Person</memberOf>
</instance>

WSML/XML representation

Legend:

Automatic transformation
Human-written mapping
Approach to Mapping #1

• Transformation between XML as defined in WSDL and the XML syntax for a target WSMO ontology
  – Using XSLT or other XML transformation language

• Advantage
  – People already know XSLT

• Disadvantage
  – XML syntax of WSML does not reflect data structure, the XSLT becomes complex
Approach to Mapping #2

- Map directly between XML and WSML instances
  - Using a mapping language specific for this task

- Advantage
  - Specific language most natural (optimized for this)

- Disadvantage
  - Yet another mapping language
Approach to Mapping #3

- Create WSMO Ontology from Schema in WSDL
  - Define mappings from conceptual framework for XML Schema to WSMO Ontology metamodel
  - Generate ontology
  - Create set of executable mapping rules for data instances

- Advantages
  - Uses WSMO data mediation
  - In simplest case no manually-created mapping required

- Disadvantage
  - Ontology-level mapping may be complex
Creating the Mappings

Step 1
WSMO Ontology Metamodel
Create Mappings at conceptual level
XML Schema Conceptual Framework

Step 2
WSMO Ontology
Create ontology using rules based on conceptual mappings
XML Schema

Step 3
WSML Data Instance
Lifting and lowering using rules based on transformation in step 2
XML Data Instance
Creating the Mappings Explained

• Define a mapping between the XML Schema Conceptual Model to the WSMO Ontology Metamodel

• Automatically create a WSMO ontology from specific XML Schema

• Create the bidirectional mappings rules to be used for the transformation between XML instances and WSMO instances.
  – Created at the same time as the generation of the WSMO ontology from an XML Schema
  – These mapping are completely derived from the actions described in the first two bullet points
Creating the Mappings
Example Scenario

- Semantic service description designer with task of providing a description for the Amazon service
- Only consider in terms of data grounding
- Two scenario use cases
  - No mediation required
  - Mediation required
Creating the Mappings: Use Case 1

- The generated ontology is sufficient for designer’s needs
- Mapping rules to get from instances of WSMO to instances of XML and vice-versa are created automatically
Creating the Mappings: Use Case 2

- Designer wishes to use a specific book ontology
- Generated ontology + rules created as before
- Additional data mediation needs to be defined (using existing ontology mapping tools)
Grounding Illustrated

1. Create WSMO description
2. Map XML schema to WSML Ontology
3. Create Mapping Rules
4. Add mapping rules to WSMO description

unnecessary in simple case
Some Discussion Points & Next Steps

- XSLT is powerful but does not take account of semantics
- Conceptual mapping offers better opportunity for reuse

- Todos
  - Formalising the mappings
  - Extending the mappings
  - Definition of how the mappings should be executed
Grounding WSMO Choreography

- Choreography representation in WSMO
  - States (made up of concepts) and transitions
- Concept modes
  - Some concepts represent in or out messages
  - In, out, shared
- Grounding property
  - Specifies a set of URIs grounding that concept
  - URIs point to WSDL In, Out or Fault messages
- URIs for identifying messages in WSDL 2.0
  ```
  http://example.com/#wsdl.interfaceMessageReference(PrinterInterface/print/In)
  ```

- WSDL → WSMO – manual (with tool support)
- WSMO → WSDL – auto generation of WSDL
  - In case WSDL doesn’t exist yet
WSMO Grounding – Summary

• Links from WSMO to the WSDL and XML Schema world
• Needed to describe an existing WSDL service
• Three steps in approach
  – Define mappings from metamodel of XML Schema to that of WSMO
  – Use the mappings to generate WSMO ontologies
  – Also generate mapping rules that can be applied at runtime to lift and lower data instances
• Simple scenarios need no data mediation
• Choreography grounded to WSDL messages
  – Can go to underlying space instead
Part I – summary and conclusions

• WSMO - a conceptual model for SWS and a basis for SWS languages and SWS execution environments; More needs to be done with respect to Web service behavior modeling
• WSML is a language for modeling of Semantic Web Services; based on the WSMO; WSML is a Web language:
  – IRIs for object identification
  – XML datatypes
• WSML is based on well-known logical formalisms: Description Logics, Logic Programming, and Frame Logic
• WSML - syntax has two parts:
  – Conceptual modeling
  – Arbitrary logical expressions
• WSML - XML and RDF syntaxes for exchange over the Web
• WSMO Discovery – a framework for SWS Discovery
• WSMO Grounding – top-down approach meets the bottom-up real world; under development
Part II: WSMO enabled systems and tools; hands-on sessions
# Part II - Agenda

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:00 – 15:00</td>
<td><strong>Web Service Execution Environment (WSMX)</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Internet Reasoning Service (IRS-III)</strong></td>
</tr>
<tr>
<td>15:00 – 15:30</td>
<td><strong>Coffee Break</strong></td>
</tr>
<tr>
<td>15:30 – 17:00</td>
<td><strong>WSMO Studio and WSMT</strong></td>
</tr>
<tr>
<td></td>
<td><strong>IRS Hands-on session</strong></td>
</tr>
</tbody>
</table>
Web Service Execution Environment (WSMX)

• Introduction, Background and motivation
• Structural architecture
• Dynamic behaviour
• Future plans
WSMX Introduction

- Software framework for runtime binding of service requesters and service providers
- WSMX interprets service requester’s goal to
  - discover matching services
  - select (if desired) the service that best fits
  - provide mediation (if required)
  - make the service invocation
- Is based on the conceptual model provided by WSMO
- Has a formal execution semantics
- SO and event-based architecture based on microkernel design using technologies as J2EE, Hibernate, Spring, JMX, etc.
WSMX Motivation

- Provide **middleware ‘glue’** for Semantic Web Services
  - Allow service providers focus on their business
- Provide a **reference implementation** for WSMO
  - Eat our own cake
- Provide an environment for **goal based** service discovery and invocation
  - Run-time binding of service requester and provider
- Provide a flexible **Service Oriented Architecture**
  - Add, update, remove components at run-time as needed
- Keep **open-source** to encourage participation
  - Developers are free to use in their own code

Define **formal execution semantics**
- Unambiguous model of system behaviour
WSMX Usage Scenario
WSMX Usage Scenario - P2P

• A P2P network of WSMX ‘nodes’
• Each WSMX node described as a SWS
• Communication via WSML over SOAP
• Distributed discovery – first aim
• Longer term aim - distributed execution environment
WSMX Usage Scenario - P2P
WSMX Usage Scenario - P2P
Development Process & Releases

• The development process for WSMX includes:
  – Establishing its conceptual model
  – Defining its execution semantics
  – Develop the architecture
  – Design the software
  – Building a working implementation

• Planned releases:
Design Principles

Strong Decoupling & Strong Mediation
autonomous components with mediators for interoperability

Interface vs. Implementation
distinguish interface (= description) from implementation (=program)

Peer to Peer
interaction between equal partners (in terms of control)

WSMO Design Principles == WSMX Design Principles
== SOA Design Principles
Benefits of SOA

• Better reuse
  – Build new functionality (new execution semantics) on top of existing Business Services

• Well defined interfaces
  – Manage changes without affecting the Core System

• Easier Maintainability
  – Changes/Versions are not all-or-nothing

• Better Flexibility
Service Oriented State

• The interface to the service is implementation-independent
• The service can be dynamically invoked
  – Runtime binding
• The service is self-contained
  – Maintains its own state
Messaging

- Messaging is peer-to-peer facility
- Distributed communication
  - Loosely coupled
- Sender does not need to know receiver (and vice versa)
- Asynchronous mechanism to communicate between software applications
WSMX Architecture

Service Oriented Architectures

Messaging

Application Management

Application Services

Management & Monitoring
Applications

Process Editor
Ontology Editor
Goal Editor

Management
Discovery
Process Mediation

Negotiation and Contracting
Orchestration
Planning

Data Mediation
Communication
Choreography

Reasoner
Semantic Repository
Triple Space

Message
Internet

Peer

Vertical Services

Peer
Selected Components

- Adapters
- Parser
- Invoker
- Choreography
- Process Mediator
- Discovery
- Data Mediator
- Resource Manager
Adapters

- To overcome data representation mismatches on the communication layer
- Transforms the format of a received message into WSML compliant format
- Based on mapping rules
Parser

- WSML compliant parser
  - Code handed over to wsmo4j initiative
    http://wsmo4j.sourceforge.net/
- Validates WSML description files
- Compiles WSML description into internal memory model
- Stores WSML description persistently (using Resource Manager)
Communication Mgr – Invoker

- WSMX uses
  - The SOAP implementation from Apache AXIS
  - The Apache Web Service Invocation Framework (WSIF)
- WSMO service descriptions are grounded to WSDL
- Both RPC and Document style invocations possible
- Input parameters for the Web Services are translated from WSML to XML using an additional XML Converter component.
Choreography

- Requester and provider have their own observable communication patterns
  - Choreography part of WSMO
- A choreography instance is loaded for each
  - Both requester and provider have their own WSMO descriptions
- The Choreography component examines a service’s choreography to determine next step in communication
- The Choreography component raises events for the Invoker to make actual service invocations
Process Mediator

• Requester and provider have their own communication patterns
• Only if the two match precisely, a direct communication may take place
• At design time equivalences between the choreographies’ conceptual descriptions is determined and stored as set of rules
• The Process Mediator provides the means for runtime analyses of two choreography instances and uses mediators to compensate possible mismatches
Discovery

• Responsible for finding appropriate Web Services to achieve a goal (discovery)
• Current discovery component is based on simple matching
• Advanced semantic discovery in prototypical stage
Discovery

Keyword-based with Natural Language Processing (NLP)

Coarse grained Service and Goal descriptions

Fine grained Service and Goal descriptions

{Keyword}

W1 … WL

WS

Level of Abstraction

Syntactic

Semantic („Light“)

Semantic („Heavy“)
Discovery

Keyword-based with Natural Language Processing (NLP)

Coarse grained Service and Goal descriptions

Fine grained Service and Goal descriptions

{Keyword}

W1 … WL

WS

Level of Abstraction

Syntactic

Semantic („Light“)

Semantic („Heavy“)
Data Mediator

- Ontology-to-ontology mediation
- A set of mapping rules are defined and then executed
- Initially rules are defined semi-automatic
- Create for each source instance the target instance(s)
Resource Manager

- Stores internal memory model to a data store
- Decouples storage mechanism from the rest of WSMX
- Data model is compliant to WSMO API
- Independent of any specific data store implementation i.e. database and storage mechanism
System Entry Points

Legend:
- WSMX components
- External entities
- Execution Flow
- Usage

storeEntity:
- Service Provider
  - Editor
  - Communication Manager (Requester Side)
    - WSML
  - Service Repository
  - Parser

receiveGoal:
- Service Requester
  - Adapter
  - Communication Manager (Requester Side)
    - WSML
  - Service Repository
  - Parser
  - Matchmaker
  - Data Mediator
  - Selector

receiveMessage:
- Service Requester
  - Adapter
  - Communication Manager (Requester Side)
    - WSML
  - Service Repository
  - Parser
  - Data Mediator
  - SOAP
  - Service Provider

WSML
WSMX components
External entities
Execution Flow
Usage
Define “Business” Process

1. Discover Web Services → Start
2. Create Choreography → Created
3. Discover Services
4. Mediate Data
5. Mediate Data
6. Return Mediated Data
7. Return Mediated Data
8. Return Web Services
9. Check Choreography → Confirmed
10. Call Invoker
11. Confirmed
12. End
Generate Wrappers for Components
Context Data

- Discover Web Services
- Start
- Create Choreography
- Created
- Discover Services
- Mediate Data
- Mediate Data
- Return Mediated Data
- Return Mediated Data
- Return Web Services
- Check Choreography
- Confirmed
- Call Invoker
- Confirmed
- End

- Choreography object
  Mediated objects, Web Services entities
- Errors
  Exceptions
Event-based Implementation
Execution Semantics

Request to discover Web services.
Execution Semantics

Goal expressed in WSML is sent to WSMX System Interface

WSMT – Web Services Modelling Toolkit

WSMX Monitor  WSMX Manager  WSML Editor  Choreography Editor  Mediator Editor

Administration Framework Interface

WSMX Manager

WSMX Manager Core

CM Wrapper  RM Wrapper  Parser Wrapper  Discovery Wrapper  Selector Wrapper  DM Wrapper  PM Wrapper  Choreography Wrapper

Interface  Interface  Interface  Interface  Interface  Interface  Interface  Interface

Communication Manager  Resource Manager  Parser  Discovery  Selector  Data Mediator  Process Mediator  Choreography Mediator

Invoker  Receiver  Grounding

WSMO Objects  Non WSMO

Resource Manager Interface  Reasoner Interface

Reasoner Interface

Component Wrapper

New Component
Execution Semantics

Com. M. implements the interface to receive WSML goals
Execution Semantics

WSMT – Web Services Modelling Toolkit

WSMX Management  WSMX Monitor  WSML Editor  Choreography Editor  Mediator Editor

WSMX Manager

WSMX Manager Core

CM Wrapper  RM  Parser

Discovery Wrapper  Selector Wrapper  DM Wrapper  PM Wrapper  Choreography Wrapper

Communication Manager

Interface

Selector

Data Mediator  Process Mediator  Choreography Mediator

Resource Manager Interface

WSMO Objects  Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component
Execution Semantics
Execution Semantics
Execution Semantics

Core is notified that choreography instance has been created.

WSMT – Web Services Modelling Toolkit

WSMX Manager

WSMX Manager Core

WSMT Editor
Choreography Editor
Mediator Editor

WSMX Manager

Service Requesters

Service Providers

Web Service 1
Web Service 2
...
Web Service p

Back-End Application
Agent acting on behalf of service requester

Adapter 1
Adapter 2
...
Adapter n

Data and Communication Protocols Adapters

Resource Manager Interface

Resource Manager Interface

WSMO Objects

Non WSMO

Reasoner Interface

Reasoner

Component Wrapper

Interface

New Component
WSMX – System Interface

WSMX Manager

WSMX Manager Core

Administration Framework Interface

Data and Communication Protocols Adapters

Adapter 1

Adapter 2

Adapter n...

Grounding

CM Wrapper

Communication Manager Interface

Invoker

RM Wrapper

Data Manager Interface

Parser

Discovery Wrapper

Selector

Selector Wrapper

DM Wrapper

Data Mediator Interface

PM Wrapper

Process Mediator Interface

Choreography

Choreography Wrapper

Choreography Interface

WSML Editor

WSMX Monitor

WSM Management

WSML Objects

Non WSMO

Reasoner Interface

Reasoner

New Component

Component Wrapper

Execution Semantics
Execution Semantics
Execution Semantics
Execution Semantics
Execution Semantics
Execution Semantics

WSMT – Web Services Modelling Toolkit

WSMX Manager Core

WSMX Manager

WSMX

WSMX Managment WSMX Monitor WSML Editor Choreography Editor Mediator Editor

Service Requesters

Agent acting on behalf of service requester

Service Providers

Web Service 1
Web Service 2
Web Service p

Back-End Application

Data and Communication Protocols Adapters

Adapter 1
Adapter 2
Adapter n

Choreography instance for goal requestor is checked for next steps.
Execution Semantics
Execution Semantics

Set of Web Service descriptions expressed in WSML sent to adapter.
Execution Semantics

Set of Web Service descriptions expressed in requester's own format returned to goal requester.
WSMX Usage Scenario - P2P

- Complete the functionality for all the boxes
WSMX Conclusions

• Conceptual model is WSMO
• End to end functionality for executing SWS
• Has a formal execution semantics
• Real implementation
• Open source code base at SourceForge
• Event-driven component architecture
• Growing functionality - developers welcome 😊
WSMX @ Sourceforge.net

Project: Web Services Execution Environment: Summary

The Web Services Execution Environment (WSMX) is an execution environment for dynamic matchmaking, selection, mediation, invocation and interoperation of Semantic Web Services.

- Development Status: v. 3 - Alpha
- Intended Audience: Developers, Science/Research
- License: MIT License
- Programming Language: Java
- Topic: Distributed Computing

Project UNIX name: wsmx
Registered: 2004-06-29 13:45
Activity Percentile (last week): 37.66%

Latest File Releases

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
<th>Date</th>
<th>Notes / Monitor</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>toolkit</td>
<td>WSMX v0.1</td>
<td>March 16, 2006</td>
<td>-</td>
<td>Download</td>
</tr>
<tr>
<td>wsmx-components</td>
<td>WSMX Components 0.1.6</td>
<td>January 31, 2005</td>
<td>-</td>
<td>Download</td>
</tr>
<tr>
<td>wsmx-core</td>
<td>WSMX Core 0.01</td>
<td>July 26, 2004</td>
<td>-</td>
<td>Download</td>
</tr>
</tbody>
</table>

Public Areas

- Project Home Page
- Supporters of this project / Make a donation

Latest News

WSMX v0.1 Released
March 16, 2006 13:17
[Read More/Comment]
IRS-III:
A framework and platform for building Semantic Web Services
IRS-III

- IRS-III: The Internet Reasoning Service is an infrastructure for publishing, locating, executing and composing *Semantic Web Services*
- Internet Reasoning Service (IRS-III):
  - System overview
  - Demonstration
Design Principles

- Ontological separation of User and Web Service Contexts
- Capability Based Invocation
- Ease of Use
- One Click Publishing
- Agnostic to Service Implementation Platform
- Connected to External Environment
- Open
- Complete Descriptions
- Inspectable
- Interoperable with SWS Frameworks and Platforms
Features of IRS-III (1/2)

- Based on Soap messaging standard
- Provides Java API for client applications
- Provides built-in brokering and service discovery support
- Provides capability-centred service invocation
Features of IRS-III (2/2)

• Publishing support for variety of platforms
  – Java, Lisp, Web Applications, Java Web Services

• Enables publication of ‘standard code’
  – Provides clever wrappers
  – One-click publishing of web services

• Integrated with standard Web Services world
  – Semantic web service to IRS
  – ‘Ordinary’ web service
IRS-III Architecture

LispWeb Server

IRS-III Server

WSMO Library

Browser

SOAP

INVOCATION HANDLER

Invoking Client

Publishing Clients

Publishing Platforms

Java API

SOAP

Web Service

Java Code

Web Application

WSMX

WSO-III Studio

Browser

Java Code

Web Application

OCML

WSO-III Studio
European Travel Scenario
European Travel Demo
IRS-III/WSMO differences

• Underlying language OCML
• Goals have inputs and outputs
• IRS-III broker finds applicable Web Services via Mediators
  – Used mediator within WS capability
  – Mediator source = Goal
• Web Services have inputs and outputs ‘inherited’ from goal descriptions
• Web Service selected via assumption (in capability)
WSMO Studio

- Integrated Service Environment for WSMO
- [http://www.wsmostudio.org](http://www.wsmostudio.org)
- Provide easy to use GUI for various WSMO tasks
  - Working with ontologies
  - Creating WSMO descriptions: goals, services, mediators
  - Creating WSMO centric orchestration and choreography specifications
  - Import (export) from (to) various formats
  - Front-end for ontology and service repositories
  - Front-end for runtime SWS environments (WSMX, IRS-III)
Requirements for an ISE

- Modular design
  - Different users need to customise the functionality in a specific way
  - Easier to maintain (e.g. ship new versions and bugfixes)
  - More suitable for 3rd party contributions
- Extensibility
  - SWS is an emerging domain
  - It is difficult to specify requirements and functionality affront
- Architecture based on open standards
  - Lowers the cost of adopting / integrating a tool
  - 3rd party extensions and improvements are more likely to occur
- Flexible licensing
  - An Open Source licence improves the adoption rate
WSMO Studio

• Java based implementation
• Open Source core
  – LGPL
  – 3rd party contributors are free to choose their respective licensing terms
• Modular design
  – an Eclipse based plug-in architecture
• Extensible
  – 3rd parties may contribute new functionality (plug-ins) or modify existing functionality
WSMO Studio architecture
WSMT

• The Web Services Modeling Toolkit (WSMT) is a collection of tools for Semantic Web services implemented in the Eclipse framework.

• Three different collections of tools:
  – WSML Perspective for creating and managing WSML descriptions
  – Data Mediation Perspective for creating mappings that can be used for runtime instance transformation
  – WSMX Perspective for managing and monitoring the Web Service Execution Environment (WSMX)
WSML Perspective

- The WSML Perspective is comprised of a number of tools that provide support for WSML engineering tasks

<table>
<thead>
<tr>
<th>Core components</th>
<th>Editors</th>
<th>Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSML</td>
<td>WSMO Visualizer</td>
<td>WSMO Reasoner</td>
</tr>
<tr>
<td>Reasoner</td>
<td>WSMO Text Editor *</td>
<td>Problems</td>
</tr>
<tr>
<td>Converter</td>
<td>Eclipse System - TE</td>
<td>Outline *</td>
</tr>
<tr>
<td>WSMO4J</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Not available in WSMT v1.2
WSMX Perspective

- The WSMX Perspective can be used to monitor the components deployed in the WSMX System and where necessary modify their behaviour.
WSMT

- WSMT is available from: http://wsmt.sourceforge.net
- Current licensing model is GPL
- Licensing Model is currently being revised to give more flexibility
- WSMT will be integrated with Distributed Ontology Management Environment (DOME) in the coming months.
- WSMT toolset is complementary to that in WSMO Studio
Semantic Web Services
Hands-On

Introduction
SWS Creation & Usage Steps

1. Create a **Goal** description
   - Define capability
   - Define pre- and post-conditions
2. Create a **WG-Mediator** description
   - Source = goal
   - Possibly add a mediation service
3. Create a **Web Service** description
   - Used-mediator = WG-Mediator above
4. Import and add Lisp function (or SOAP endpoint etc.) in **Choreography Grounding**
5. **Publish** against Web Service description
6. **Invoke** Web Service by ‘achieve goal’
Multiple WS for goal

• Each Web Service has a WG-Mediator in used-mediator slot
  – some **Web Services may share a Mediator**
• Each Web Service carries **Assumptions** over its inputs
  – in IRS these are defined in OCML
• Web Service chosen for Goal according to these Assumptions = **Capability-Driven Invocation**
Defining a Mediation Service

- **Mediation Goal** defines necessary transformation between Goal and Service
  - Mediation Goal input roles are a subset of Goal’s input roles
- **WG-Mediator** has Mediation Goal as source
- **Mediation Service** uses this mediator
- WG-Mediator linking Goal and its Service has Mediation Goal in ‘uses mediation service’ slot
IRS-III Hands On Task

• Develop an application for the European Travel scenario based on SWS. The application should support a person booking a train ticket between 2 European cities at a specific time and date

• Create Goal, Web service and Mediator WSMO descriptions in WSMO Studio

• Import definitions into IRS-III ‘european-travel-service-descriptions’ ontology

• In IRS Browser ground (to Lisp functions) and add (OCML) assumptions to services, then deploy.

• Invoke the web services
Tutorial Setup

WSMO Studio

IRS Server (3000)

Domain Models

IRS Lisp Publisher

IRS-III Browser & Editor

Travel Services (3001)
Travel Related Knowledge Models
Key Classes, Relations, Instances

is-in-country <city> <country> e.g.
(is-in-country berlin germany) -> true

(student <person>) -> true, for john matt michal
(business-person <person>) -> true, for liliana michael
Goals

1- Get train timetable
   – Inputs: origin and destination cities (city), date (date-and-time, e.g. (18 4 2004))
   – Output: timetable (string)

2- Book train
   – Inputs: passenger name (person), origin and destination cities, departure time-date (list-date-and-time, e.g. (20 33 16 15 9 2004))
   – Output: booking information (string)
Services

• 1 service available for goal 1
  – No constraints

• 6 services available for goal 2
  – As a provider write the constraints applicable to the services to satisfy the goal (assumption logical expressions)

• 1 wg-mediator mediation-service
  – Used to convert time in list format to time in universal format
Service constraints

• Services 2-5
  – Services for (origin and destination) cities in determined countries

• Service 4-5
  – Need a mediation service to map goal time-date to service time-date

• Services 6-7
  – Services for students or business people in Europe
Available Functions (1/3)

1- **get-train-times**
   
   *paris london (18 4 2004)*
   
   "Timetable of trains from PARIS to LONDON on 18, 4, 2004
   
   5:18
   
   …23:36"

2- **book-english-train-journey**
   
   *christoph milton-keynes london (20 33 16 15 9 2004)*
   
   "British Rail: CHRISTOPH is booked on the 66 going from MILTON-KEYNES to LONDON at 16:49, 15, SEPTEMBER 2004. The price is 169 Euros."

3- **book-french-train-journey**
   
   *sinuhe paris lyon (3 4 6 18 8 2004)*
   
   "SNCF: SINUHE is booked on the 511 going from PARIS to LYON at 6:12, 18, AUGUST 2004. The price is 27 Euros."
Available Functions (2/3)

4- book-german-train-journey

cristoph berlin frankfurt 3304251200

"First Class Booking German Rail (Die Bahn): CHRISTOPH is booked on the 323 going from BERLIN to FRANKFURT at 17:11, 15, SEPTEMBER 2004. The price is 35 Euros."

5- book-austrian-train-journey

sinuhe vienna innsbruck 3304251200

"Austrian Rail (OBB): SINUHE is booked on the 367 going from VIENNA to INNSBRUCK at 16:47, 15, SEPTEMBER 2004. The price is 36 Euros. "
Available Functions (3/3)

6- book-student-european-train-journey

John London Nice (3 4 6 18 8 2004)
"European Student Rail Travel: JOHN is booked on the 916 going from LONDON to NICE at 6:44, 18, AUGUST 2004. The price is 94 Euros."

7- book-business-european-train-journey

Liliana Paris Innsbruck (3 4 6 18 8 2004)
"Business Europe: LILIANA is booked on the 461 going from PARIS to INNSBRUCK at 6:12, 18, AUGUST 2004.
The price is 325 Euros."

8- mediate-time (lisp function) or

JavaMediateTime/mediate (java)
(9 30 17 20 9 2004)
3304686609
Example: Goal
Example: Mediator
Example: Service
Example: Grounding and Publishing
Closing, Outlook, References, Acknowledgements
Tutorial Wrap-up

• The targets of the presented tutorial were to:
  – understand aims & challenges within Semantic Web Services
  – understand WSMO approach to Semantic Web Services
    • WSML
    • WSMO Discovery
    • WSMO Grounding
  – present WSMX and IRS - future Web Service based IT middlewares
    • design and architecture
    • components design

=> you should now be able to correctly assess WSMO technologies and utilize these for your future work
Beyond WSMO

- Although WSMO is one of the main initiatives on Semantic Web services, there are also other initiatives in the area: OWL-S, SWSL, WSDL-S
- Semantic Web Services Interest Group
  - Interest group founded at W3C to discuss issues related to Semantic Web Services (http://www.w3.org/2002/ws/swsig/)
  - Standardization Working Group in starting phase
- SWSI: International initiative to push toward a standardization of SWS (http://www.swsi.org)
- Semantic Web services are entering the main stream
  - UDDI is adopting OWL for semantic search
  - WSDL 2 will contain a mapping to RDF
  - The use of semantics is also discussed in the context of standards for WS Policies
SWSI (www.swsi.org)

- SWSI (Semantic Web Services Initiative) is becoming the point of synthesis of the SWS activity around the World
- SWSI includes many participants belonging to both academy and industry from the US and Europe
- SWSI is composed of two committees
  - SWSL which is expected to produce a language for Semantic Web services
  - SWSA which is expected to describe the architectural requirements for Semantic Web services
- OWL-S and WSMO are two main inputs, but contributions include IRS, Meteor-S
Semantics in the Main Stream

- Many WS standardization groups are realizing that they need to add semantic representation
- **UDDI v.next**
  - UDDI v.next is the new version of UDDI
  - UDDI TC has decided to use OWL as a standard language for the representation of business taxonomies
  - OWL-based inference will be used to improve WS search
- **Web Service Description Language v2**
  - The WSDL working group at W3C has decided to add an RDF mapping to WSDL 2
  - The RDF mapping may effectively provide a standard grounding mechanism for OWL-S and WSMO
- **Web Services policies proposals** require a significant amount of inference
  - There have been proposals to use OWL or SWRL as basic languages
  - Or to provide a mapping to semantic Web languages
References WSMO

• The central location where WSMO work and papers can be found is WSMO Working Group: http://www.wsmo.org

• WSMO languages
  – WSML Working Group http://www.wsml.org

• Web Service Execution Environment WSMX
  – WSMX working group : http://www.wsmx.org
  – WSMX open source can be found at:
    https://sourceforge.net/projects/wsmx/
References WSMO


The central location where WSMX work, papers, and software can be found is the WSMX working group homepage: http://www.wsmx.org.

The main documents are:
- Conceptual Model (http://www.wsmo.org/2004/d13/d13.1/v0.3/)
- Architecture (http://www.wsmo.org/TR/d13/d13.4/v0.2/)
- Implementation: open source at http://sourceforge.net/projects/wsmx
- Documentation (http://www.wsmo.org/TR/d22/v0.2/)
- Execution Semantics (http://www.wsmo.org/TR/d13/d13.2/)
- WSMX Toolkit (http://www.wsmo.org/TR/d9/d9.1/v0.2/)

Further Readings:
References IRS III


These papers and software downloads can be found at: http://kmi.open.ac.uk/projects/irs
Acknowledgements

The WSMO work is funded by the European Commission under the projects ASG, DIP, Knowledge Web, SEKT, SWWS, AKT and Esperonto; by Science Foundation Ireland under the DERI-Lion project; and by the Austrian government under the FIT-IT program.

IRS development is funded by the European Commission under the DIP project, and formerly IBROW, and by the UK EPSRC under the AKT project, and formerly MIAKT.